



Semi-two-dimensional partitioning for parallel sparse matrix-vector multiplication

Enver Kayaaslan, Bora Uçar, Cevdet Aykanat

► To cite this version:

Enver Kayaaslan, Bora Uçar, Cevdet Aykanat. Semi-two-dimensional partitioning for parallel sparse matrix-vector multiplication. PCO2015 (IPDPSW), May 2015, Hyderabad, India. pp.1125–1134. hal-01159692

HAL Id: hal-01159692

<https://inria.hal.science/hal-01159692>

Submitted on 3 Jun 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Semi-two-dimensional partitioning for parallel sparse matrix-vector multiplication

Enver Kayaaslan, Bora Uçar
{INRIA, CNRS} and LIP (UMR 5668: ENS Lyon,
CNRS, UCBL, Université de Lyon, INRIA),
46 allée d'Italie, 69364, Lyon, France
Email: {enver.kayaaslan, bora.ucar}@ens-lyon.fr

Cevdet Aykanat
Bilkent University
06800, Bilkent, Ankara, Turkey
Email: aykanat@cs.bilkent.edu.tr

Abstract—We propose a novel sparse matrix partitioning scheme, called semi-two-dimensional (s2D), for efficient parallelization of sparse matrix-vector multiply (SpMV) operations on distributed memory systems. In s2D, matrix nonzeros are more flexibly distributed among processors than one dimensional (rowwise or columnwise) partitioning schemes. Yet, there is a constraint which renders s2D less flexible than two-dimensional (nonzero based) partitioning schemes. The constraint is enforced to confine all communication operations in a single phase, as in 1D partition, in a parallel SpMV operation. In a positive view, s2D thus can be seen as being close to 2D partitions in terms of flexibility, and being close 1D partitions in terms of computation/communication organization. We describe two methods that take partitions on the input and output vectors of SpMV and produce s2D partitions while reducing the total communication volume. The first method obtains optimal total communication volume, while the second one heuristically reduces this quantity and takes computational load balance into account. We demonstrate that the proposed partitioning method improves the performance of parallel SpMV operations both in theory and practice with respect to 1D and 2D partitionings.

Keywords—Sparse matrix-vector multiplication, matrix partitioning.

I. INTRODUCTION

We consider efficient parallelization of sparse matrix-vector multiply (SpMV) operations of the form $\mathbf{y} \leftarrow \mathbf{A}\mathbf{x}$ in distributed memory systems. Such a parallelization requires the data (the nonzeros of \mathbf{A} , the entries of the input vector \mathbf{x} and the entries of the output vector \mathbf{y}) to be partitioned among available processors. According to the distribution of matrix elements, the data partitions are called one-dimensional (1D), or two-dimensional (2D). In 1D partitions, the matrix is partitioned rowwise or columnwise. In 2D partitions, the nonzeros in a row or in a column can be assigned to more than one processor. With 1D partitions, the parallel SpMV algorithms have a single communication phase in which either \mathbf{x} - or \mathbf{y} -vector entries are communicated; whereas with 2D partitions, the parallel SpMV algorithms have two communication phases—first one is on \mathbf{x} and the second one is on \mathbf{y} . In general, 2D partitions have more flexibility to be exploited in optimizing the parallel performance. On the other hand, 1D partitions has the advantage of confining the communication to a single phase. For a fair treatment of parallel SpMV, the associated matrix partitioning methods and a survey of

methods the reader is referred to elsewhere [1, Chapter 4][5]. In this work, we propose a modification of the 2D-partition-based parallel SpMV algorithm, where the communications on \mathbf{x} and \mathbf{y} are handled in a single phase as in the 1D-partition-based parallel SpMV algorithm. This modification necessitates a special, 2D distribution of the nonzeros of the input matrix. We describe the characteristics of the required distribution and propose efficient heuristics to obtain them.

A. Context

Consider the fine-level computations $y_i \leftarrow y_i + a_{ij} \times x_j$ that compose $\mathbf{y} \leftarrow \mathbf{A}\mathbf{x}$. The multiplication here is performed by the processor which holds the nonzero a_{ij} . The addition is performed by the same processor locally, and then it is possibly sent to the processor which is responsible to accumulate results on y_i . The standard SpMV algorithm based on such a *fine-grain* partition therefore proceeds in three phases:

- 1) (Expand) Each processor sends its input-vector entries x_j to other processors that keep a nonzero a_{ij} .
- 2) Each processor performs the fine-multiplications for each a_{ij} kept by the processor and accumulates to its own partial-result \bar{y}_i , that is, $\bar{y}_i \leftarrow \bar{y}_i + a_{ij} \times x_j$.
- 3) (Fold) Each processor receives the partial-results \bar{y}_i from other processors that keep a nonzero a_{ij} and aggregates them to obtain its output-vector entries y_i .

Step 1 *Expand* and Step 3 *Fold* are communication phases. When data partition is 1D, one of these communication phases vanishes; e.g., in the columnwise partitioning, a_{ij} and x_j are kept together by the same processor, for each a_{ij} , and there is no communication in the expand phase. The 1D partitionings lead to reasonable performance in the parallel SpMV when the input matrix possesses an approximately regular number of nonzeros per row/or column. However, this is often not the case, and the parallel performance deteriorates due to poor computational balance and excessive use of bandwidth and latency. The 2D partitions come to rescue in these cases by not restricting nonzero partition on a column or row; but the parallel SpMV algorithm has both the *expand* and the *fold* phases and, hence, possibly increased number of messages.

Let us take a close look at the fine-level computations performed in the parallel SpMV, while assuming a data distribution. We can divide them into four groups: (i) both

x_j and y_i are local; (ii) x_j is local, y_i is non-local; (iii) y_i is local, x_j is non-local; (iv) both x_j and y_i are non-local. Here, x_j (similarly y_i) is called “local”, if it is kept by the same processor that holds a_{ij} , and “non-local” otherwise. First, we note that the computational group (i) can be performed locally with no communication. Second, the group (ii) is independent of the expand phase, as x_j entries are already present at the time of computation. However, the computations in (ii) should be carried out before the fold phase. Third, the computational group (iii) is suspended until the processor receives x_j entries, yet the partial-results are not to be communicated since y_i entries are local. The computational group (iv) requires special attention. Since both x_j and y_i entries are non-local, the computation is linked with both of the communication phases. The fine-multiplication with nonzeros in this group is the only reason that the expand phase has to precede the fold phase. Therefore, we can perform the two communication phases concurrently if the computational group (iv) has no nonzeros.

B. Contributions and organization

We propose *semi-two-dimensional* (s2D) data partitioning, which restricts each nonzero a_{ij} to be together with either x_j or y_i . By this restriction, we guarantee that the computational group (iv) is empty, and thus, the two communication phases can be fused into a single one. The s2D partitioning has more flexibility on the distribution of data than that of 1D methods, and requires single communication phase in parallel SpMV. This necessitates a reorganization in the parallel SpMV algorithm and new partitioning methods which are presented in Sections III and IV. Some recent work, summarized in Section V, recognizes the inadequacy of 1D methods in modern applications and develop related alternatives. We perform some comparisons with them and present results in Section VI.

II. BACKGROUND

We collect material that we use later. Throughout the paper K denotes the number of processors; data is partitioned into K parts, each of which corresponds to a processor.

A. 1D (rowwise) partitioning

In the 1D rowwise partitioning scheme, the sparse matrix \mathbf{A} is partitioned in compliance with the output-vector \mathbf{y} . That is, each processor P_k holds an output-subvector $\mathbf{y}^{(k)}$ and a submatrix $\mathbf{A}^{(k)}$ such that $a_{ij} \in \mathbf{A}^{(k)}$ only if $y_i \in \mathbf{y}^{(k)}$. Thus, we can permute and write \mathbf{A} as $\mathbf{A} = [\mathbf{A}_{1*}^T; \mathbf{A}_{2*}^T; \dots; \mathbf{A}_{K*}^T]^T$. Processor P_k holds the row-stripe $\mathbf{A}^{(k)} = \mathbf{A}_{k*} = [\mathbf{A}_{k1}, \mathbf{A}_{k2}, \dots, \mathbf{A}_{kK}]$ and is responsible to compute $\mathbf{y}^{(k)} \leftarrow \mathbf{A}^{(k)} \mathbf{x}$. The input-vector \mathbf{x} can be partitioned arbitrarily. Then, the input- and output-vector partition induces the block structure on \mathbf{A} such that $a_{ij} \in \mathbf{A}_{\ell k}$ only if $y_i \in \mathbf{y}^{(\ell)}$ and $x_j \in \mathbf{x}^{(k)}$.

B. The Dulmage-Mendelsohn decomposition

Let \mathbf{B} be an arbitrary rectangular matrix, and $\hat{m}(\mathbf{B})$ and $\hat{n}(\mathbf{B})$ represent the number of nonempty rows and the number

of nonempty columns in \mathbf{B} , respectively. Then, we build the Dulmage-Mendelsohn (DM) decomposition [10] on \mathbf{B} as,

$$\hat{\mathbf{B}} = \begin{bmatrix} \mathbf{H} & \mathbf{X} & \mathbf{Z} \\ 0 & \mathbf{S} & \mathbf{Y} \\ 0 & 0 & \mathbf{V} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \hat{\mathbf{B}} & 0 \\ 0 & 0 \end{bmatrix},$$

where $\hat{\mathbf{B}}$ is the nonzero block of \mathbf{B} . Here \mathbf{H} , \mathbf{S} , and \mathbf{V} refer to horizontal, square, and vertical blocks of \mathbf{B} , respectively. As their names suggest, we have $\hat{m}(\mathbf{H}) < \hat{n}(\mathbf{H})$, $\hat{m}(\mathbf{S}) = \hat{n}(\mathbf{S})$, and $\hat{m}(\mathbf{V}) > \hat{n}(\mathbf{V})$. This block triangular decomposition is based on the canonical decomposition of bipartite graphs induced by maximum matchings [10], [15]. This implies that $\hat{m}(\mathbf{H}) + \hat{m}(\mathbf{S}) + \hat{n}(\mathbf{V})$ is the minimum number of rows and columns required to cover all nonzeros.

III. PARALLEL SPMV FOR S2D PARTITIONING

In this section, we introduce s2D partitioning. We also show how to modify the parallel SpMV in accordance with this partitioning to obtain a single phase of communication.

Let \mathbf{A} be an $m \times n$ sparse matrix, and \mathbf{x} and \mathbf{y} be dense input- and output-vectors of size n and m , respectively. In s2D partitioning, the matrix \mathbf{A} and the vectors \mathbf{x} and \mathbf{y} are partitioned into K parts, where each processor P_k holds submatrix $\mathbf{A}^{(k)}$, and subvectors $\mathbf{x}^{(k)}$ and $\mathbf{y}^{(k)}$ such that $a_{ij} \in \mathbf{A}^{(k)}$ only if $x_j \in \mathbf{x}^{(k)}$ or $y_i \in \mathbf{y}^{(k)}$. Then, \mathbf{A} can be permuted and written as

$$\mathbf{A} = \mathbf{A}^{(1)} + \mathbf{A}^{(2)} + \dots + \mathbf{A}^{(K)},$$

where

$$\mathbf{A}^{(k)} = \begin{bmatrix} & & & \mathbf{A}_{1k}^{(k)} & & \\ & & & \mathbf{A}_{2k}^{(k)} & & \\ & & & \vdots & & \\ \mathbf{A}_{k1}^{(k)} & \mathbf{A}_{k2}^{(k)} & \dots & \mathbf{A}_{kk}^{(k)} & \dots & \mathbf{A}_{kK}^{(k)} \\ & & & \vdots & & \\ & & & \mathbf{A}_{Kk}^{(k)} & & \end{bmatrix}, \quad (1)$$

or each $1 \leq k \leq K$. The off-diagonal block $\mathbf{A}_{\ell k}$ induced by $\mathbf{y}^{(\ell)}$ and $\mathbf{x}^{(k)}$, where $\ell \neq k$, can now be written as

$$\mathbf{A}_{\ell k} = \mathbf{A}_{\ell k}^{(k)} + \mathbf{A}_{\ell k}^{(\ell)}. \quad (2)$$

An s2D partition is therefore can be considered as a 2D partition where $\mathbf{A}_{qr}^{(k)} \neq \mathbf{0}$ only if $q = k$ or $r = k$.

In Section I, we presented a categorization of computational work into four groups. The key observation is that an s2D partition admits only the computational groups having either x_j or y_i local with respect to a_{ij} . One can fuse the expand and the fold communications in a single phase, if the task of computational group (ii) “ x_j is local, y_i is non-local” is relocated prior to the expand phase. In the modified parallel SpMV, each processor P_k executes the following steps:

- 1) (**Precompute**) For each $\ell \neq k$;
 - a) Compute $\mathbf{y}_k^{(\ell)} \leftarrow \mathbf{A}_{\ell k}^{(k)} \mathbf{x}^{(k)}$,
 - b) Form vector $\hat{\mathbf{y}}_k^{(\ell)}$, which contains only those entries of $\mathbf{y}_k^{(\ell)}$ corresponding to nonzero rows in $\mathbf{A}_{\ell k}^{(k)}$.

- 2) **(Expand-and-Fold)** For each $\ell \neq k$;
 - a) Form vector $\hat{\mathbf{x}}_\ell^{(k)}$, which contains only those entries of $\mathbf{x}^{(k)}$ corresponding to nonzero columns in $\mathbf{A}_{\ell k}^{(\ell)}$,
 - b) Send vector $[\hat{\mathbf{x}}_\ell^{(k)}, \hat{\mathbf{y}}_k^{(\ell)}]$ to processor P_ℓ .
- 3) **(Compute)** Compute the output-subvector $\mathbf{y}^{(k)}$ as

$$\mathbf{y}^{(k)} \leftarrow \sum_{\ell \neq k} \mathbf{A}_{k\ell}^{(k)} \hat{\mathbf{x}}_\ell^{(\ell)} + \mathbf{A}_{kk} \mathbf{x}^{(k)} + \sum_{\ell \neq k} \hat{\mathbf{y}}_\ell^{(k)}.$$

This modified parallel SpMV is composed of three phases, where two of them are computational. The first phase (**Precompute**) computes the partial-results associated with nonzeros a_{ij} such that x_j is local and y_i is non-local. The processor P_k computes the partial-result vector $\hat{\mathbf{y}}_k^{(\ell)} = [\bar{y}_i : a_{ij} \in \mathbf{A}_{\ell k}^{(\ell)}]$ to be sent to processor P_ℓ for each $\ell \neq k$. The second phase (**Expand-and-Fold**) is the one that performs the communication. The partial-result vector $\hat{\mathbf{y}}_k^{(\ell)}$ is accompanied with the vector $\hat{\mathbf{x}}_\ell^{(k)}$ of input-vector entries required by P_ℓ , that is $\hat{\mathbf{x}}_\ell^{(k)} = [x_j : a_{ij} \in \mathbf{A}_{\ell k}^{(\ell)}]$. Then, P_k sends the combined vector $[\hat{\mathbf{x}}_\ell^{(k)}, \hat{\mathbf{y}}_k^{(\ell)}]$ to P_ℓ . In the third phase (**Compute**), the associated output-subvector $\mathbf{y}^{(k)}$ is computed by aggregating the partial-results of three kinds of sources: $\mathbf{A}_{k\ell}^{(\ell)}$ ($\ell \neq k$), \mathbf{A}_{kk} , and $\mathbf{A}_{k\ell}^{(\ell)}$ ($\ell \neq k$). The first one is the nonzero block $\mathbf{A}_{k\ell}^{(\ell)}$ where $\ell \neq k$, and $\hat{\mathbf{x}}_\ell^{(k)}$ refers to the input-subvector received during the *Expand-and-Fold* phase from processor P_ℓ . Here $\hat{\mathbf{x}}_\ell^{(k)}$ is required to perform fine-multiplications associated with nonzeros in $\mathbf{A}_{k\ell}^{(\ell)}$. The second one is the diagonal block \mathbf{A}_{kk} , which is also expected to contain the bulk of computation for efficiency. The third source type is the nonzero block $\mathbf{A}_{k\ell}^{(\ell)}$ where $\ell \neq k$. The associated fine-multiplications have already been performed (in *Precompute*), and the partial results are received (in *Expand-and-Fold*). Thus, P_k just accumulates those partial-results $\hat{\mathbf{y}}_\ell^{(k)}$ to the output-subvector $\mathbf{y}^{(k)}$.

We make three observations on this modified parallel SpMV. First, a message is communicated from P_k to P_ℓ only when $[\hat{\mathbf{x}}_\ell^{(k)}, \hat{\mathbf{y}}_k^{(\ell)}]$, and equivalently $\mathbf{A}_{\ell k}$, is nonempty. Notice that the input- and output-vector partitions are the only factor that determines whether $\mathbf{A}_{\ell k}$ is empty or not. Therefore, the communication patterns for s2D and 1D partitions are identical whenever they have vector partitions in common. Second, P_k sends the input-vector entries $\hat{\mathbf{x}}_\ell^{(k)}$ and the partial-results $\hat{\mathbf{y}}_k^{(\ell)}$ required by P_ℓ in a single message. Thus, s2D partitioning not only simplifies the parallel SpMV by coupling the two communication phases, but can also reduce the latency overhead with respect to 2D methods. The last observation is that the modification has no effect on being a generalization of row-parallel SpMV. That is, when the partition is 1D rowwise, the precompute phase is not needed, and the single communication boils down to the expand phase.

We now give the formulation for the communication volume (bandwidth) as done in similar studies [18]. P_k sends an input-vector entry $x_j \in \mathbf{x}^{(k)}$ to P_ℓ , whenever the corresponding column in $\mathbf{A}_{\ell k}^{(\ell)}$ has a nonzero. Thus, the communication volume $\lambda_{k \rightarrow \ell}^x$ incurred by the input-vector entries is

$$\lambda_{k \rightarrow \ell}^x = |\hat{\mathbf{x}}_\ell^{(k)}| = |\{x_j \in \mathbf{x}^{(k)} : a_{ij} \in \mathbf{A}_{\ell k}^{(\ell)}\}| = \hat{n}(\mathbf{A}_{\ell k}^{(\ell)}).$$

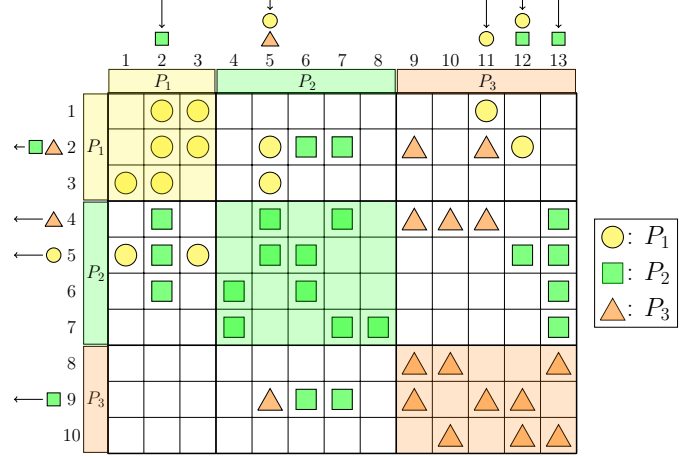


Fig. 1: A sample s2D partitioning of 10×13 sparse matrix. The shapes above the columns and left to the rows indicate communication requirements due to the corresponding column or row; e.g., P_2 sends $[x_5, \bar{y}_2]$ to P_1 in a single message, where input-vector entry x_5 is needed due to $a_{2,5}$ and partial result \bar{y}_2 is precomputed by P_2 as $\bar{y}_2 = a_{2,6} \times x_6 + a_{2,7} \times x_7$.

Likewise, P_ℓ receives a partial result $\bar{y}_i \in \hat{\mathbf{y}}_k^{(\ell)}$ from P_k , whenever the corresponding row in $\mathbf{A}_{\ell k}^{(k)}$ has a nonzero. Then, the communication volume $\lambda_{k \rightarrow \ell}^y$ due to the partial results is

$$\lambda_{k \rightarrow \ell}^y = |\hat{\mathbf{y}}_k^{(\ell)}| = |\{y_i \in \mathbf{y}^{(\ell)} : a_{ij} \in \mathbf{A}_{\ell k}^{(k)}\}| = \hat{m}(\mathbf{A}_{\ell k}^{(k)}),$$

and the total communication volume $\lambda_{k \rightarrow \ell}$ from P_k to P_ℓ is

$$\lambda_{k \rightarrow \ell} = \hat{n}(\mathbf{A}_{\ell k}^{(\ell)}) + \hat{m}(\mathbf{A}_{\ell k}^{(k)}). \quad (3)$$

Figure 1 displays a 10×13 sparse matrix \mathbf{A} along with a sample 3-way s2D partitioning on it. Notice that each nonzero is assigned to either its row part or column part, e.g., $a_{2,5}$ is assigned to its row part P_1 , whereas $a_{2,6}$ is assigned to its column part P_2 . Also notice that nonzeros of diagonal blocks are assigned to their corresponding parts. P_1 requires x_5 from P_2 for fine-level multiplication with $a_{2,5}$ and $a_{3,5}$. On the other hand, the partial result \bar{y}_2 is also needed to be communicated from P_2 to P_1 , where P_2 has precomputed $\bar{y}_2 = a_{2,6} \times x_6 + a_{2,7} \times x_7$. Therefore, in the *Expand-and-Fold* phase, P_2 sends $[x_5, \bar{y}_2]$ to P_1 within a single packet. In the figure, a circle, rectangle, or triangle upon the columns indicate that the respective \mathbf{x} -vector entry is required for the corresponding part. For instance, P_2 is the only processor that requires x_{13} , since P_2 is the only one that has nonzeros in that column. Similarly, a shape left to rows indicate that the partial results of the respective \mathbf{y} -vector entry is communicated from the corresponding part, e.g., P_1 sends a partial result \bar{y}_5 to P_2 due to $a_{5,1}$ and $a_{5,3}$. Here, the total communication volume $\lambda_{3 \rightarrow 2}$ from P_3 to P_2 is 3, as $\hat{n}(\mathbf{A}_{2,3}^{(2)}) = 2$ and $\hat{m}(\mathbf{A}_{2,3}^{(3)}) = 1$.

IV. A COMBINATORIAL METHOD FOR S2D PARTITIONING

We propose an s2D partitioning method with two steps. The first step accepts an input- and output-vector partition into K parts. In the second step, we build an s2D partition with respect to that vector partition. The choice of vector

partitioning in the first step is related to how an s2D partition is built upon in the second step. In the proposed approach, 1D rowwise partitioning is the most relevant one to obtain a vector partition, since we initially assign nonzeros together with their corresponding output vector entries, and then apply a coarse-grain refinement.

We formalize s2D nonzero partitioning problem as follows.

Problem 1: Given a K -way partition on \mathbf{x} and a K -way partition on \mathbf{y} , find a K -way partition on \mathbf{A} such that for each nonzero a_{ij} , $\pi(a_{ij}) \in \{\pi(y_i), \pi(x_j)\}$, where $\pi(\cdot)$ refers to the owner of a data element.

Let each processor P_k have an input-subvector $\mathbf{x}^{(k)}$ and an output-subvector $\mathbf{y}^{(k)}$. This vector partition induces a K by K block structure on \mathbf{A} , where

$$\mathbf{A}_{\ell k} = \{a_{ij} : y_i \in \mathbf{y}^{(\ell)}, x_j \in \mathbf{x}^{(k)}\}.$$

Our aim is to build two blocks $\mathbf{A}_{\ell k}^{(k)}$ and $\mathbf{A}_{\ell k}^{(\ell)}$ for each off-diagonal block $\mathbf{A}_{\ell k}$, which is induced by the input- and output-vector partition, such that

$$\mathbf{A}_{\ell k} = \mathbf{A}_{\ell k}^{(k)} + \mathbf{A}_{\ell k}^{(\ell)}.$$

We first show how to find such an s2D partition with optimal total communication volume

$$\lambda = \sum_{\ell, k} \lambda_{k \rightarrow \ell}, \quad (4)$$

for the given input- and output-vector partition. Then, we take the load balance into account, and propose a heuristic to obtain an s2D partition that reduces both the maximum computational load and the total communication volume.

A. Optimizing the total communication volume

As the blocks are disjoint, the total communication volume λ given in (4) can be minimized by optimizing the communication volume of each off-diagonal block independently. Below, we describe how to minimize the communication volume $\lambda_{k \rightarrow \ell}$ of an off-diagonal block $\mathbf{A}_{\ell k}$.

We first obtain the DM decomposition of $\mathbf{A}_{\ell k}$ as

$$\hat{\mathbf{A}}_{\ell k} = \begin{bmatrix} \mathbf{H}_{\ell k} & \times & \times \\ 0 & \mathbf{S}_{\ell k} & \times \\ 0 & 0 & \mathbf{V}_{\ell k} \end{bmatrix}, \quad \mathbf{A}_{\ell k} = \begin{bmatrix} \hat{\mathbf{A}}_{\ell k} & 0 \\ 0 & 0 \end{bmatrix}.$$

Based on this DM decomposition, we build two blocks $\mathbf{A}_{\ell k}^{(\ell)}$ and $\mathbf{A}_{\ell k}^{(k)}$ as

$$\hat{\mathbf{A}}_{\ell k}^{(\ell)} = \begin{bmatrix} \times & \times \\ \mathbf{S}_{\ell k} & \times \\ 0 & \mathbf{V}_{\ell k} \end{bmatrix}, \quad \hat{\mathbf{A}}_{\ell k}^{(k)} = \mathbf{H}_{\ell k}, \quad (5)$$

where $\hat{\mathbf{A}}_{\ell k}^{(\ell)}$ and $\hat{\mathbf{A}}_{\ell k}^{(k)}$ refer to the nonzero block of $\mathbf{A}_{\ell k}^{(\ell)}$ and $\mathbf{A}_{\ell k}^{(k)}$, respectively. Due to the formula given in (3), we have

$$\lambda_{k \rightarrow \ell} = \hat{n}(\mathbf{A}_{\ell k}^{(\ell)}) + \hat{n}(\mathbf{A}_{\ell k}^{(k)}). \quad (6)$$

The DM decomposition suggests that $\lambda_{k \rightarrow \ell}$, when computed as in (6), is the minimum that can be achieved by distributing nonzeros of $\mathbf{A}_{\ell k}$ into two blocks $\mathbf{A}_{\ell k}^{(k)}$ and $\mathbf{A}_{\ell k}^{(\ell)}$.

B. Bi-objective optimization

The computational load of a processor is proportional to the number of nonzeros assigned to it. Hence, for an s2D partition as in (1), the computational load of a given processor P_k is

$$|\mathbf{A}^{(k)}| = \sum_{\ell \neq k} |\mathbf{A}_{k\ell}^{(k)}| + |\mathbf{A}_{kk}| + \sum_{\ell \neq k} |\mathbf{A}_{\ell k}^{(k)}|. \quad (7)$$

Then, we compute the maximum computational load \widetilde{W} as

$$\widetilde{W} = \max_k |\mathbf{A}^{(k)}|.$$

Our aim is to minimize both the total communication volume λ and the maximum computational load \widetilde{W} . As we need to handle two different objectives, we consider the following problem: Given an upper bound W_{lim} to the maximum computational load, find an s2D partition with respect to the given input- and output-vector partition such that $\widetilde{W} \leq W_{\text{lim}}$ and λ is minimum.

We follow a coarse-grain approach that adapts the one used in optimizing the total communication volume. We consider two alternatives for each off-diagonal block $\mathbf{A}_{\ell k}$:

$$(A1) \quad \mathbf{A}_{\ell k}^{(\ell)} = \mathbf{A}_{\ell k}, \text{ and } \mathbf{A}_{\ell k}^{(k)} = \mathbf{0},$$

$$(A2) \quad \mathbf{A}_{\ell k}^{(\ell)} = \mathbf{A}_{\ell k} - \mathbf{H}_{\ell k} \text{ and } \mathbf{A}_{\ell k}^{(k)} = \mathbf{H}_{\ell k} \text{ (cf. (5)).}$$

One obtains

$$\begin{aligned} \lambda_{k \rightarrow \ell}^{A1} &= \hat{n}(\mathbf{A}_{\ell k}), \\ \lambda_{k \rightarrow \ell}^{A2} &= \hat{n}(\mathbf{A}_{\ell k} - \mathbf{H}_{\ell k}) + \hat{n}(\mathbf{H}_{\ell k}), \end{aligned}$$

with the first and second alternatives, respectively. Recall that $\lambda_{k \rightarrow \ell}^{A2}$ is the minimum that can be achieved. Then, we reduce our problem to choose an alternative for each off-diagonal block so that λ is minimum while $\widetilde{W} \leq W_{\text{lim}}$. This problem is NP-complete (contains Knapsack as a special case), and we resort to heuristics.

We propose Algorithm 1. In this algorithm, we start with choosing the alternative (A1) for all off-diagonal blocks. This initial partitioning coincides with the 1D rowwise partitioning of \mathbf{A} with respect to given input and output vector partition. Then, we go over the off-diagonal blocks multiple times, to alternate the choices from (A1) to (A2). Any alternation is final—we do not alternate our choice back to (A1). The iterations over the off-diagonal blocks are in decreasing order of the associated reductions in the communication volume. This reduction, which is denoted by $\lambda_{\ell k}^-$, is equal to the number of columns minus the number of rows of the corresponding horizontal block. That is, for an off-diagonal block $\mathbf{A}_{\ell k}$,

$$\lambda_{\ell k}^- = \lambda_{k \rightarrow \ell}^{A2} - \lambda_{k \rightarrow \ell}^{A1} = \hat{n}(\mathbf{H}_{\ell k}) - \hat{n}(\mathbf{H}_{\ell k}).$$

At the moment of deciding to alternate the choice of an off-diagonal block or not, our only consideration is the maximum computational load. That is, we check if the alternation cause the maximum computational load exceed the limit. Notice that this algorithm assumes the initial maximum computational load $\widetilde{W} = \max_k |\mathbf{A}_{k*}|$ is less than W_{lim} . However, this may not necessarily be the case. Therefore, we implement the algorithm such that we check if $W_k + |\mathbf{H}_{\ell k}|$ is smaller than the current maximum load \widetilde{W} instead of W_{lim} , whenever $\widetilde{W} > W_{\text{lim}}$ during the execution of the algorithm.

Algorithm 1 s2DPARTITIONING(\mathbf{A} , $\{\mathbf{x}^{(k)}\}$, $\{\mathbf{y}^{(k)}\}$)

Input \mathbf{A} : sparse matrix.
Input $\{\mathbf{x}^{(k)}\}$: K -way input-vector partition.
Input $\{\mathbf{y}^{(k)}\}$: K -way output-vector partition.
Output $\{\mathbf{A}^{(k)}\}$: K -way nonzero partition

```
for each processor  $P_k$  do
   $W_k \leftarrow |\mathbf{A}_{k*}|$ 
  for each off-diagonal block  $\mathbf{A}_{\ell k} \neq \mathbf{0}$  do
    Choose (A1) for  $\mathbf{A}_{\ell k}$ 
   $\tilde{W} \leftarrow \max_k W_k$ ,  $flag \leftarrow \text{TRUE}$ 
  while  $flag = \text{TRUE}$  do
     $flag \leftarrow \text{FALSE}$ 
    for each off-diagonal block  $\mathbf{A}_{\ell k} \neq \mathbf{0}$  do
      ► in decreasing order of  $\lambda_{\ell k}^-$ 
      if (A1) is chosen for  $\mathbf{A}_{\ell k}$  then
        if  $W_k + |\mathbf{H}_{\ell k}| \leq \max\{\tilde{W}, W_{lim}\}$  then
          Choose (A2) for  $\mathbf{A}_{\ell k}$ 
           $W_k \leftarrow W_k + |\mathbf{H}_{\ell k}|$ 
           $W_\ell \leftarrow W_\ell - |\mathbf{H}_{\ell k}|$ 
           $\tilde{W} \leftarrow \max_k W_k$ ,  $flag \leftarrow \text{TRUE}$ 
  return  $\{\mathbf{A}^{(k)}\}$ 
```

V. RELATED RECENT WORK AND COMPARISON

We summarize some recent work on partitioning methods for parallel SpMV in modern applications.

Kuhlemann and Vassilevski [12] recognize the need to reduce the number of messages in parallel SpMV operations with matrices corresponding to scale-free graphs. They present methods to embed the given graph in a bigger one to reduce the number of messages. The gist of the method is to split a vertex into a number of copies (the number is determined with a simple calculation to limit the maximum number of messages per processor). In such a setting, the SpMV operations with the matrix associated with the original graph, $\mathbf{y} \leftarrow \mathbf{A}\mathbf{x}$, is then cast as triple sparse matrix vector products of the form $\mathbf{y} \leftarrow \mathbf{Q}^T(\mathbf{B}(\mathbf{Q}\mathbf{x}))$, much like in multigrid solvers. This original work can be extended to other matrices (not necessarily symmetric, nor square) by recognizing the triplet product as a communication on \mathbf{x} for duplication (for the columns that are split), communication of \mathbf{x} vector entries (duplicates are associated with different destinations), multiplication, and as a communication on the output vector (for the rows that are split) to gather results. This exciting extension requires further analysis, which is out of our scope in this paper.

Boman et al. [2] propose a 2D partitioning method obtained by post-processing a 1D partition. Given a 1D partition among P processors, the method maps the $P \times P$ block structure to a virtual mesh of size $P_r \times P_c$ and reassigns the off-diagonal blocks so as to limit the number of messages per processor by $P_r + P_c$. The post-processing is fast, and hence the method is as nearly efficient as a 1D partitioning method. However, the communication volume and the load balance obtained in the 1D partitioning phase are disturbed (this does not seem to cause problems for their applications), and the method does not have any means to control the perturbation.

Pelt and Bisseling [14] propose a model to partition sparse matrices into two parts (which then can be used recursively

to partition into any number of parts). The essential idea has two steps. First, the nonzeros of a given matrix \mathbf{A} are split into two different matrices (of the same size as the original matrix), say $\mathbf{A} = \mathbf{A}_r + \mathbf{A}_c$. Second, \mathbf{A}_r and \mathbf{A}_c are partitioned together, where \mathbf{A}_r is partitioned rowwise, and \mathbf{A}_c is partitioned columnwise. As all nonzeros of \mathbf{A} are in only one of \mathbf{A}_r or \mathbf{A}_c , the final result is a two-way partitioning of the nonzeros of \mathbf{A} . The resulting partition on \mathbf{A} achieves load balance and reduces the total communication volume by the standard hypergraph partitioning techniques. Consider the following interpretation of this approach. First split the matrix as before $\mathbf{A} = \mathbf{A}_r + \mathbf{A}_c$. Then, build a composite hypergraph model [17] using the enriched column-net hypergraph for \mathbf{A}_r and the row-net hypergraph for \mathbf{A}_c and amalgamate the vertices as required. Now consider a K -way partition of the composite model. In this partition, a column in \mathbf{A}_c is assigned to a part which holds the corresponding \mathbf{x} vector entry. Similarly, a row in \mathbf{A}_r is assigned to a part which holds the corresponding \mathbf{y} vector entry. Therefore, the output from this partition can be decoded as an s2D partition. A few notes are necessary. First, in a preliminary set of experiments we tested the use of composite hypergraph models and performed a standard hypergraph partitioning on it (instead of using splits and iterative refinement at every recursive bisection step as done by Pelt and Bisseling). Our tests agreed with the reported results [14] in that the medium-grain approach improved upon the fine-grain [6] method. Second, the use of composite models enable obtaining symmetric vector partitions (that is the partitions on \mathbf{x} and \mathbf{y} are the same), while exactly encoding the total communication volume. To the best of our understanding, a symmetric vector partitioning is not always possible with the original approach of Pelt and Bisseling using the standard hypergraph partitioning methods.

Two-dimensional partitioning methods that bound the maximum number of messages per processor, such as the checkerboard [5], [7] and orthogonal recursive bisection [18] based, have been used in modern applications [19], [21], sometimes without graph/hypergraph partitioning [20]. In almost all cases, inadequacy of 1D partitioning schemes are confirmed.

All these previous work assumes the standard SpMV multiplication algorithm based on expanding \mathbf{x} -vector entries, performing multiplies with matrix entries, and folding \mathbf{y} -vector entries. Compared to all these previous work, ours has therefore a distinctive characteristic. We search for a special 2D partition on the matrix elements in which a nonzero is assigned to a processor holding the associated input or output vector entry. This is essential for the proposed SpMV algorithm in which the expand and fold messages are combined. The implication is that there is a single communication phase (all the previous work based on 2D partitions has two communication phase) as is the case when 1D partitions are used; yet there is also a greater flexibility to reduce the communication volume than the 1D method. In the next section, we compare the proposed method with the well-known 1D and 2D partitioning schemes, together with the schemes proposed by Boman et al. [2] and Pelt and Bisseling [14].

name	n	nnz	d_{avg}	d_{max}	application
crystk02	13965	968583	69.4	81	materials problem
turon_m	189924	1690876	8.9	11	structural engineering
trdheim	22098	1935324	87.6	150	structural engineering
c-big	345241	2340859	6.8	19578	non-linear optimization
ASIC_680k	682862	2638997	3.9	388488	circuit simulation
3dtube	45330	3213618	70.9	2364	structural engineering
pkustk12	94653	7512317	79.4	4146	structural engineering
pattern1	19242	9323432	484.5	6028	optimization problem

TABLE I: Properties of the test matrices used in comparison with 1D and 2D methods. The sparse matrices are displayed in increasing order of their nonzero counts (nnz). Here, d_{avg} and d_{max} are the average and the maximum number of nonzeros of rows, respectively.

VI. EXPERIMENTAL RESULTS

We evaluate the performance of the s2D partitioning (s2D) in two setups. In the first one (results are in Section VI-A), we compare it with the 1D rowwise [3] (1D) and the 2D fine-grain [6] (2D) partitioning methods with $K \in \{16, 64, 256\}$ processors on a set of eight sparse matrices from the University of Florida Matrix (UFL) Collection [9]. We also measure SpMV timings on a Cray XE6 system, and we observe that s2D displays considerable improvement on the overall execution times with respect to both 1D and 2D, on average. However, as also observed in related studies (e.g., [11], [16]), the latency overhead dominates when K gets larger. Therefore, we build the second setup, which mainly focuses on the partitioning methods that bound latency overhead of a processor. For the second setup (results are in Section VI-B), we consider another set of eight sparse matrices, which contain high degree rows, from UFL Collection [9] and Stanford Network Analysis Platform (SNAP) [13]. These matrices naturally cause very high number of messages sent by a processor, and thus parallel computation with bounded latency is required. We measure the performance results with $K \in \{256, 1024, 4096\}$. To achieve a bounded latency for each processor, one common approach is the Cartesian partitioning, which can be obtained with hypergraph models (2D-b) [5], [7]. In this study, we follow a technique similar to the post-processing of Boman et al. [2] to manipulate an s2D partitioning in order to achieve a partitioning (s2D-b) with bounded maximum latency. We will observe that the proposed method s2D-b uniformly outperforms the state-of-the-art methods 2D-b and 1D-b in both the load balance and the total communicational volume. Finally, we compare the proposed heuristic for the s2D partitioning with the adapted version of the medium-grain partitioning Pelt and Bisseling [14], which is denoted with s2D-mg.

A. Comparison with 1D and 2D methods

Table I displays the properties of the matrices used to evaluate the performance of the proposed method s2D with respect to 1D and 2D methods, which refer to the 1D rowwise and the 2D fine-grain partitioning, respectively. We measure the performances with $K = \{16, 64, 256\}$ processors in terms of the partitioning qualities as well as their SpMV timings. We used a Cray XE6 system to measure the parallel performance. At each node, the system has two 16-core AMD processors

with 2.3 GHz clock frequency, and 32 GB memory. The interconnection network is a high speed 3D torus so called CRAY Gemini. We used one core at a node.

Table II displays a comparative summary of the performance results for 1D and 2D methods, and for the proposed method, s2D. For 1D and 2D methods, we used the column-net [3] and the row-column-net [5] hypergraph models to obtain the rowwise and the fine-grain partition, respectively. We used PaToH [4] as the hypergraph partitioner in default setting with imbalance tolerance of 3%. For s2D, we used the heuristic presented in Section IV-B to build an s2D partition with respect to the vector partition induced by the rowwise partition that is used for 1D. Therefore, s2D and 1D exhibit the same communication pattern. As PaToH relies on randomization, we present the geometric averages of three runs for each partitioning instance.

We first investigate the performance in terms of the partitioning qualities. Here, we give all the numbers for $K = 256$, though the statements are general. As seen in the table, s2D method reduces the total communication volume for each partitioning instance with respect to 1D. We observe that this reduction is correlated with the skewness of the nonzero distribution on rows. The matrices trdheim and ASIC_680k exhibit very low and very high skewed nonzero distributions, and correspondingly, lead to 2% and 96% reductions in the communication volume, respectively. Also note that the computational load balance is either better than or equal to that of 1D. For instance, s2D reduces the load imbalance from 120% to 16%, while improving the communication volume by 42% for c-big. Notice that 2D method achieves a good computational balance (0.1%, on average), however the communication latency requirement is consistently larger than that of 1D and s2D methods, and displays approximately 60% increase in both total and maximum number of messages, on average. Also note that s2D method exhibits communication volume comparable to 2D, e.g., the average scaled volumes are 0.52 and 0.43 for s2D and 2D, respectively.

We now consider the parallel SpMV times. We display the speedup values with respect to a serial implementation. For six of the matrices, we observe the best speedup under s2D column, whereas on the matrices pkustk12 and trdheim, 1D and 2D, respectively, exhibit the best performance (167.9 and 89.2 speedups, respectively). Moreover, s2D displays superior timing performance, on average, for all $K \in \{16, 64, 256\}$.

The table shows that the latency overhead of communication is a dominating factor (e.g., ASIC_680k achieves 2.4 and 9.2 speedups for 1D and s2D, respectively, on 256 processors) on the performance, especially when K gets larger. In particular, the matrices with dense rows (e.g., c-big, ASIC_680k, and pattern1) tend to load a processor with very high number of messages to be received (or sent), in their parallel multiplication for 1D, and hence s2D. Notice that 2D method exhibit a similar performance for those matrices, since 2D implements the fine-grain partitioning method, which takes only the communication volume and the computational load balance into account. Thus, for sparse matrices with dense

name	K	1D partitioning (1D)					2D partitioning (2D)					s2D partitioning (s2D)		
		LI%	Latency avg	Latency max	λ_{1D}	Sp	LI%	Latency avg	Latency max	$\frac{\lambda}{\lambda_{1D}}$	Sp	LI%	$\frac{\lambda}{\lambda_{1D}}$	Sp
crystk02	16	1.0%	4	6	8.19e3	14.9	0.0%	8	14	1.28	14.4	1.0%	0.98	15.3
	64	1.4%	8	13	2.27e4	46.1	0.0%	15	24	1.16	30.7	1.4%	0.95	36.3
	256	2.0%	12	22	5.36e4	44.6	0.0%	21	36	1.02	37.9	2.0%	0.91	53.5
turon_m	16	0.4%	5	7	1.76e4	15.5	1.4%	8	11	0.48	15.8	0.3%	0.80	15.4
	64	0.6%	6	8	3.89e4	43.4	1.2%	9	15	0.49	44.2	0.6%	0.81	39.1
	256	0.9%	8	14	7.92e4	88.3	1.1%	13	21	0.52	117.2	0.9%	0.84	128.0
trdheim	16	0.7%	3	4	4.45e3	16.3	0.4%	5	9	1.18	15.3	0.7%	0.95	16.9
	64	0.6%	3	7	1.54e4	51.2	0.3%	7	13	1.10	36.0	0.5%	0.99	53.6
	256	0.9%	5	11	4.59e4	75.7	0.2%	11	20	0.99	89.2	0.9%	0.98	57.0
c-big	16	2.3%	14	15	6.39e4	14.9	0.1%	28	30	0.51	15.2	2.2%	0.55	18.1
	64	2.2%	29	63	9.64e4	29.9	0.2%	60	94	0.58	28.0	2.2%	0.59	47.0
	256	1.2*	34	253	1.38e5	6.9	0.2%	64	285	0.62	9.3	16.6%	0.58	7.7
3dtube	16	0.9%	5	9	1.73e4	16.5	0.0%	10	14	1.03	15.9	0.8%	0.92	16.6
	64	0.8%	6	17	3.88e4	62.1	0.0%	12	21	1.17	49.6	0.8%	0.98	60.9
	256	12.9%	8	36	9.32e4	100.6	0.0%	17	48	1.13	74.7	12.9%	0.80	123.0
ASIC_680k	16	1.4*	14	15	7.87e5	4.0	0.3%	22	29	0.01	16.9	41.4%	0.02	12.9
	64	8.4*	24	57	7.95e5	3.7	0.3%	35	89	0.01	35.8	41.7%	0.02	37.0
	256	36.7*	25	225	8.06e5	2.4	0.3%	33	278	0.03	12.2	43.1%	0.04	9.2
pkustk12	16	0.9%	4	14	4.57e4	16.7	0.6%	13	22	0.30	16.3	0.9%	0.93	16.7
	64	1.2%	9	25	1.39e5	64.7	0.2%	20	58	0.24	53.8	1.2%	0.91	63.5
	256	8.3%	18	40	2.39e5	167.9	0.2%	22	149	0.32	45.7	8.3%	0.55	162.9
pattern1	16	2.2%	12	15	6.07e4	20.2	0.2%	30	30	0.39	18.8	2.0%	0.69	20.8
	64	2.2%	23	63	1.14e5	55.4	0.4%	54	106	0.51	63.6	2.2%	0.70	66.8
	256	2.2%	42	252	2.79e5	23.2	0.5%	63	308	0.41	23.9	2.2%	0.85	23.1
geomean	16	1.9%	6	10	3.34e4	13.7	0.1%	13	18	0.36	16.0	1.5%	0.51	16.4
	64	2.6%	10	23	7.09e4	35.5	0.1%	20	39	0.40	41.2	1.8%	0.54	49.2
	256	10.6%	15	54	1.38e5	34.4	0.1%	25	85	0.43	37.2	4.8%	0.52	43.5

TABLE II: Performance results of s2D with respect to 1D and 2D, on $K \in \{16, 64, 256\}$ processors, in terms of the load imbalance (LI%), where x^* refers to $100x\%$, the total communication volume (λ_{1D} is the exact volume for 1D, and the volume of 2D and s2D are given as normalized to λ_{1D}), and the average and maximum number of messages sent by a processor. The communication pattern, and thus, the latency values of s2D are same with that of 1D. The speed-up values (Sp) are given with respect to serial implementation and the best of each rows is shown in bold, and the best of each matrix is underlined.

name	Sp (best of 1D, 2D, s2D)	HP-based Cartesian partitioning (2D-b)				
		LI%	Latency avg	Latency max	$\frac{\lambda}{\lambda_{1D}}$	Sp
crystk02	53.5 (s2D)	2.7%	13	19	1.18	62.2
turon_m	128.0 (s2D)	3.3%	9	16	1.24	79.5
trdheim	89.2 (2D)	2.1%	7	14	1.30	72.1
c-big	9.3 (2D)	4.4%	23	30	1.39	85.4
ASIC_680k	12.2 (2D)	1.6*	21	29	1.24	37.9
3dtube	123.0 (s2D)	3.6%	11	18	1.27	113.2
pkustk12	167.9 (1D)	3.7%	15	26	1.16	186.3
pattern1	23.9 (2D)	14.0%	24	30	0.91	214.0
geomean	43.5 (s2D)	6.3%	14	22	1.20	92.2

TABLE III: Comparison of Cartesian partitioning that is based on hypergraph partitioning (2D-b) with the best of 1D, 2D, and s2D methods for $K = 256$ in terms of speed-up (Sp) with respect to the serial implementation. The communication volume is given as normalized to that of 1D. The load imbalance (LI%) of ASIC_680k, i.e., 1.6*, refers to $\sim 160\%$.

rows, partitionings that bound the maximum latency are essential to achieve efficient parallelization.

Table III compares the previously discussed methods 1D, 2D, and s2D, with Cartesian partitioning 2D-b, called checkerboard [5], [7]. As seen in the table, in five of the test cases, 2D-b demonstrates better speedups than all 1D,

2D, and s2D. Notice that the improvement is much more pronounced for those matrices with dense rows. For instance, 2D-b improves speedup from 12.2 (for 2D) to 37.9 despite its large load imbalance of 160%.

B. Comparison with recent work

1) *Cartesian methods*: Table IV shows the properties of the matrices that we used to assess the proposed s2D-b method for $K \in \{256, 1024, 4096\}$. Two of the matrices (c-big and ASIC_680k) exist also in the previous setup. Another two are obtained through SNAP [13], where com-Youtube is the online social network of Youtube, and rmat_20 is a R-MAT [8] graph (edges made undirected) with parameters $a=0.57$, $b=c=0.19$, and $d=0.05$. The matrices in this set have some dense rows (see d_{\max} in the table), e.g., the maximum number of nonzeros of a row of c-big is approximately 6% and 1% of the total number of columns and nonzeros, respectively, and ins2 contains a row that is full of nonzeros.

We propose s2D-b in order to bound the maximum latency requirement at the expense of extra communication phase, similar to that of Boman et al. [2]. Unlike the standard Cartesian partitionings, s2D-b enables that the input-vector entries and the partial output-vector results are communicated together at each communication phase. We first compare

name	n	nnz	d_{avg}	d_{max}	description
boyd2	466316	1500397	3.2	93263	optimization
lp1	534388	1643420	3.1	249644	optimization
c-big	345241	2340859	6.8	19579	non-linear opt.
ASIC_680k	682862	2638997	3.9	388489	optimization
ins2	309412	2751484	8.9	309413	circuit sim.
com-Youtube	1157827	5975248	5.2	28755	Youtube social
rajat30	643994	6175244	9.6	454747	circuit sim.
rmat_20	1048576	8174570	7.8	23716	Graph500 ben.

TABLE IV: Properties of the test matrices with dense rows to be used in discussion of the proposed $s2D-b$ method. The sparse matrices are displayed in increasing order of their nonzero counts (nnz). Here, d_{avg} and d_{max} refer to the average and maximum nonzero counts of rows, respectively.

$s2D-b$ with $s2D$ in order to show the interplay between the latency and the bandwidth requirements. Table V displays the performance of 1D, $s2D$ and $s2D-b$ in the set of matrices with dense rows, where 1D is also included in order to provide a unified perspective. As seen in the table, 1D method fails to achieve an acceptable partitioning quality, in terms of either the computational load, or the communication latency, or bandwidth. The maximum number of nonzeros of a row is $1.07e5$, and total number of nonzeros is $3.26e6$. Thus, it is theoretically impossible to achieve perfect balance for larger than 30 processors with 1D methods, on average, and the balance degenerates linearly with the increasing number of processors. Owing to the increased flexibility of nonzeros in partitioning, $s2D$ method exhibit reasonable performance on the computational load. For instance, we observe 83.8% load imbalance, on average for $K = 4096$ of $s2D$, which is equivalent to performing ~ 2229 processors with perfect balance. As $s2D-b$ employs the same nonzero partition with $s2D$, we observe such improvement on the computational load balance for $s2D-b$, as well.

Table V displays a significant reduction in the total communication volume with $s2D$ when compared to 1D methods. For instance, on average, we observe a reduction of 95% and 80%, for $K = 256$ and 4096, respectively. As seen in the table, $s2D-b$ increases communication volume with respect to $s2D$. Nevertheless, the communication volume of $s2D-b$ is still significantly less than 1D, e.g., 94% and 76% reduction, for $K=256$ and 4096, respectively, on average.

In terms of the communication latency, 1D, and equivalently $s2D$, perform in such a way that the maximum number of messages sent by a processor is in the order of K , e.g., 3579 messages for $K = 4096$, on average. By the virtue of the block layout of processors and the two-phase communication of $s2D-b$, we observe that the maximum latency requirement is in the order of \sqrt{K} , e.g., 90 messages for $K = 4096$, on average, for $s2D-b$.

Table VI compares $s2D-b$ with the state-of-the-art methods 2D-b [7] and 1D-b [2]. For $s2D-b$, we use the same partitions with the ones displayed in Table V. Moreover, we construct 1D-b on the same vector partition with that of $s2D-b$. All methods exhibit comparable performance for both maximum and average latency requirement. For instance, the total and the maximum number of messages is $19 \times K$ and 90

for 2D-b (cf. $18 \times K$ and 90 for $s2D-b$, and equivalently for 1D-b), respectively, with $K = 4096$, on average. Therefore, we compare the methods only in terms of the computational load balance and the total communication volume.

We first discuss the computational load balance. As seen in Table VI, 2D-b displays 75.1% load imbalance, and we observe that the imbalance increases linearly with K , e.g., 202.8% and 508.2% for $K=1024$ and $K=4096$, respectively, on average. The load balance is much more dramatic for 1D-b. For instance, we observe 841.1% load imbalance with 4096 processors, on average, for 1D-b. The proposed method $s2D-b$, on the other hand, distributes the computational load in considerably more balance than 2D-b and 1D-b, e.g., 83.8% load imbalance with 4096 processors, on average.

In terms of the total communication volume, $s2D-b$ is the clear winner for this set of matrices, where 2D-b and 1D-b perform comparable to each other. The high bandwidth requirement of 2D-b and 1D-b methods is inevitable due to the high degree rows. The method $s2D-b$ alleviates this problem by sending the precomputed partial y -vector results along with the x -vector entries. As seen in the table, $s2D-b$ performs uniformly better than 2D-b and 1D-b methods for real-life matrices, and, on average, displays 84% reduction in the communication volume compared to 2D-b for $K=4096$.

As a result, we observe that on the matrices with high degree rows/columns, the proposed method $s2D-b$ improves the state-of-the-art methods 1D-b and 2D-b in both the total communication volume and the computational load balance.

2) *Medium-grain partitioning*: Table VII compares the proposed method $s2D$ with $s2D-mg$. Recall that $s2D-mg$ refers to the medium-grain method [14] adapted to produce an $s2D$ partition. As seen in the table, $s2D-mg$ achieves better load balance than $s2D$. For instance, for $K = 256$ on average, $s2D$ achieves 52.3% load imbalance, whereas $s2D-mg$ displays a load imbalance of 4.8%. On the positive side, $s2D$ exhibits significantly less communication volume than $s2D-mg$, for instance, it nearly halves the bandwidth requirement for $K = 256$, on average. However, for some instances, $s2D-mg$ has better performance, such as ASIC_680k and rajat30. Nevertheless, having large number of processors is in favor of $s2D$. For instance, we observe $s2D$ reduces the communication volume by 16% for ASIC_680k when $K = 4096$, with respect to $s2D-mg$.

As seen in the table, the total latency requirement of $s2D$ is considerably less than that of $s2D-mg$ in general. On average, $s2D-mg$ and $s2D$ require $38 \times K$ and $30 \times K$ messages to be communicated, respectively, for $K = 4096$. There are cases where the improvement in the total latency requirement is much more pronounced, such as boyd2, where the total number of messages for $s2D-mg$ and $s2D$ are $33 \times K$ and $6 \times K$, respectively, for $K=4096$.

To compare, $s2D$ reduces the communication volume significantly, and exhibits better latency performance, whereas $s2D-mg$ achieves better load balance. We notice that the performance gap closes with increasing K for each criterion.

name	K	1D				s2D		s2D-b		
		LI%	Latency		λ_{1D}	LI%	$\frac{\lambda}{\lambda_{1D}}$	Latency		$\frac{\lambda}{\lambda_{1D}}$
			avg	max				avg	max	
boyd2	256	14.9*	5	239	2.30e5	33.1%	0.01	4	25	0.01
	1024	62.7*	5	945	2.40e5	32.3%	0.02	5	38	0.02
	4096	253.6*	6	3731	2.51e5	29.8%	0.09	5	66	0.10
lp1	256	41.6*	2	225	2.50e5	38.3%	0.00	2	16	0.00
	1024	169.4*	2	918	2.51e5	63.7%	0.01	2	32	0.01
	4096	680.5*	2	3984	2.57e5	1.3*	0.04	2	64	0.04
c-big	256	2.1*	34	253	1.38e5	66.1%	0.58	21	30	0.82
	1024	11.2*	24	1005	1.96e5	1.2*	0.62	23	57	0.86
	4096	47.8*	15	3563	3.16e5	1.4*	0.64	17	107	0.84
ASIC_680k	256	41.5*	25	225	8.06e5	61.4%	0.03	18	28	0.05
	1024	168.9*	21	897	8.26e5	63.4%	0.06	19	46	0.08
	4096	678.8*	15	3578	8.63e5	65.9%	0.10	15	78	0.12
ins2	256	29.0*	24	218	6.91e5	45.0%	0.01	14	29	0.01
	1024	119.1*	22	839	7.00e5	54.3%	0.03	14	56	0.04
	4096	479.2*	21	3310	7.75e5	61.4%	0.11	13	101	0.12
com-Youtube	256	1.7*	232	255	1.60e6	2.4*	0.65	30	30	0.91
	1024	9.7*	468	964	2.20e6	2.9*	0.69	60	62	0.98
	4096	41.8*	366	3306	2.83e6	3.3*	0.73	104	126	1.05
rajat30	256	17.9*	18	212	1.20e6	28.7%	0.05	14	28	0.06
	1024	74.4*	20	823	1.31e6	34.5%	0.08	18	50	0.10
	4096	300.6*	20	3237	1.48e6	42.0%	0.14	19	78	0.16
rmat_20	256	1.4%	255	255	4.53e6	34.5%	0.72	30	30	0.94
	1024	3.1*	1019	1022	5.94e6	82.0%	0.81	62	62	1.09
	4096	15.3*	1337	4011	6.88e6	76.0%	0.90	124	126	1.27
geomean	256	5.3*	26	235	6.65e5	52.3%	0.05	12	27	0.06
	1024	38.9*	32	924	7.65e5	71.7%	0.10	16	49	0.12
	4096	163.7*	30	3579	8.90e5	83.8%	0.20	18	90	0.24

TABLE V: Performance results of s2D-b with respect to 1D and s2D for $K \in \{256, 1024, 4096\}$ in terms of the load imbalance (LI%, with x^* being $100x\%$), the total communication volume (λ_{1D} is that of 1D, and those of s2D and s2D-b are given as normalized to λ_{1D}), and the average and maximum number of messages sent by a processor. The communication pattern and the latency values of s2D and 1D are the same. The load imbalance values of s2D and s2D-b are the same.

VII. CONCLUSION AND RESEARCH DIRECTIONS

We introduced semi-two-dimensional (s2D) partitioning of a sparse matrix for parallel sparse matrix vector multiply (SpMV) operations. This partitioning has more flexibility than one dimensional (rowwise or columnwise) partitioning schemes. Yet, there is a constraint which makes it less flexible than nonzero-based, two dimensional partitionings. Each nonzero is assigned to the processor that has the associated input- or output-vector entry. We modified the organization of the standard SpMV operation to take advantage of this constraint to confine all communication operations in a single phase. That is, s2D has a data distribution more general than 1D and less general than 2D, yet it enables a single phase of communication in SpMV as 1D schemes.

We proposed a two-step method to obtain an s2D partitioning. In the first step, a given input- and output-vector partition is used to impose a block structure in the input matrix. In the second step, horizontal blocks of nonzeros are carefully re-assigned to processors to optimize the total communication volume with respect to the partition imported in the first step, or traded off with the load balance. Using a suit of real-life matrices, we observed speed-ups with respect to serial execution with s2D method (49.2) compared to 1D rowwise (35.5) and 2D fine-grain (41.2), on average, using a Cray

XE6 system. We also observed that the merit of the proposed method is much more evident in matrices having rows (or columns) with a high number of nonzeros. For those matrices, we saw 97% to 85% improvement on the communication volume for 256 and 4096 processors, respectively, on average.

We achieved significant improvement on the communication volume, but the load balance was not as good as that of fine-grain (or medium-grain [14]) partitionings. This depends on the vector partition assumed in the first step of the proposed heuristic. More sophisticated heuristics that also take square and vertical blocks of off-diagonal blocks into account can be considered in the second step to mitigate this dependency. More advanced methods to find input-vector, output-vector, and nonzero partition, simultaneously can also be devised.

By making use of an idea from a recent work [2], we devised a variant (s2D-b) of the proposed s2D method, which bounds the maximum latency requirement by a number in the order of square root of the number of processors. According to our experiments, s2D-b outperformed the best alternatives (which have the same latency bound) in both load balance (e.g., from 508.2% to 83.8% with 4096 processors) and the communication volume (e.g., 82% reduction for 4096 processors). However, s2D-b requires more elaboration, particularly in terms of SpMV timings, which we leave as future work.

name	K	2D-b		1D-b [2]		s2D-b	
		LI%	λ_{2D-b}	LI%	$\frac{\lambda}{\lambda_{2D-b}}$	LI%	$\frac{\lambda}{\lambda_{2D-b}}$
boyd2	256	54.2%	4.25e5	1.8*	0.96	33.1%	0.00
	1024	2.0*	4.43e5	5.3*	0.98	32.3%	0.01
	4096	4.8*	4.55e5	13.5*	0.99	29.8%	0.05
lp1	256	2.5*	4.98e5	3.3*	0.97	38.3%	0.00
	1024	6.2*	5.00e5	6.7*	0.98	63.7%	0.00
	4096	16.2*	5.04e5	13.6*	0.99	1.3*	0.02
c-big	256	48.7%	1.92e5	84.5%	1.01	66.1%	0.59
	1024	64.1%	2.72e5	1.1*	0.98	1.2*	0.62
	4096	2.8*	4.15e5	3.1*	0.97	1.4*	0.64
ASIC_680k	256	2.0*	1.00e6	1.7*	0.99	61.4%	0.04
	1024	5.0*	1.03e6	4.4*	0.92	63.4%	0.06
	4096	11.7*	1.07e6	9.9*	0.92	65.9%	0.10
ins2	256	1.8*	1.27e6	2.9*	0.94	45.0%	0.01
	1024	4.8*	1.34e6	7.1*	0.87	54.3%	0.02
	4096	10.3*	1.40e6	19.3*	0.79	61.4%	0.07
com-Youtube	256	1.2*	2.99e6	2.4*	0.57	2.4*	0.48
	1024	1.3*	3.72e6	3.6*	0.64	2.9*	0.58
	4096	4.8*	4.51e6	13.7*	0.71	3.3*	0.66
rajat30	256	1.0*	2.24e6	3.3*	0.81	28.7%	0.03
	1024	2.8*	2.41e6	6.8*	0.82	34.5%	0.05
	4096	7.4*	2.58e6	15.9*	0.82	42.0%	0.09
rmat_20	256	3.7%	3.59e6	3.7%	0.87	34.5%	1.19
	1024	41.3%	4.97e6	51.5%	0.91	82.0%	1.31
	4096	47.8%	6.51e6	1.1*	0.94	76.0%	1.34
geomean	256	75.1%	1.03e6	1.3*	0.88	52.3%	0.04
	1024	2.0*	1.18e6	3.3*	0.88	71.7%	0.08
	4096	5.1*	1.35e6	8.4*	0.89	83.8%	0.16

TABLE VI: Performance results of s2D-b with respect to 2D-b and 1D-b [2] for $K \in \{256, 1024, 4096\}$ in terms of the load imbalance (LI%, with x^* being $100x\%$), and the communication volume (λ_{2D-b} is that of 2D-b; those of 1D-b and s2D-b are given as normalized to λ_{2D-b}).

name	K	s2D-mg [14]			s2D		
		LI%	Lat.	λ_{mg}	LI%	Lat.	$\frac{\lambda}{\lambda_{mg}}$
boyd2	256	0.1%	105	1.25e5	33.1%	5	0.01
	1024	0.4%	85	1.29e5	32.3%	5	0.04
	4096	1.0%	33	1.48e5	29.8%	6	0.16
lp1	256	9.6%	2	5.10e2	38.3%	2	0.91
	1024	9.8%	2	2.15e3	63.7%	2	0.90
	4096	10.8%	2	1.06e4	1.3*	2	1.01
c-big	256	43.2%	40	8.82e4	66.1%	34	0.90
	1024	43.5%	31	1.39e5	1.2*	24	0.88
	4096	47.5%	20	2.33e5	1.4*	15	0.87
ASIC_680k	256	13.1%	28	2.08e4	61.4%	25	1.34
	1024	13.2%	23	4.27e4	63.4%	21	1.07
	4096	13.3%	15	1.02e5	65.9%	15	0.84
ins2	256	5.0%	19	1.41e4	45.0%	24	0.53
	1024	5.1%	25	3.84e4	54.3%	22	0.62
	4096	5.7%	18	1.01e5	61.4%	21	0.87
com-Youtube	256	1.2*	243	1.49e6	2.4*	232	0.69
	1024	1.2*	552	2.00e6	2.9*	468	0.76
	4096	1.3*	411	2.54e6	3.3*	366	0.82
rajat30	256	0.8%	20	4.15e4	28.7%	18	1.31
	1024	1.1%	24	9.09e4	34.5%	20	1.21
	4096	1.3%	23	1.98e5	42.0%	20	1.07
rmat_20	256	1.0%	255	3.27e6	34.5%	255	0.99
	1024	39.5%	972	4.86e6	82.0%	1019	0.99
	4096	60.3%	1072	6.14e6	76.0%	1337	1.01
geomean	256	4.8%	39	6.54e4	52.3%	26	0.52
	1024	9.4%	50	1.24e5	71.7%	32	0.61
	4096	11.9%	38	2.42e5	83.8%	30	0.74

TABLE VII: Performance results of s2D with respect to s2D-mg [14] for $K \in \{256, 1024, 4096\}$ in terms of the load imbalance (LI%, with x^* being $100x\%$), the average number of messages (Lat.), and the total communication volume (λ_{mg} is that of s2D-mg; that of s2D is given as normalized to λ_{mg}).

ACKNOWLEDGEMENTS

Bora Uçar is supported by ANR project SOLHAR (ANR-13-MONU-0007).

REFERENCES

- [1] R. H. Bisseling, *Parallel Scientific Computation: A Structured Approach using BSP and MPI*. Oxford University Press, Oxford, UK, 2004.
- [2] E. G. Boman, K. D. Devine, and S. Rajamanickam, “Scalable matrix computations on large scale-free graphs using 2D graph partitioning,” in *Proc. International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, pp. 50:1–50:12.
- [3] Ü. V. Çatalyürek and C. Aykanat, “Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication,” *IEEE T. Parall. Distr.*, vol. 10, no. 7, pp. 673–693, 1999.
- [4] —, *PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.2*, Bilkent Univ., Dept. Computer Engineering, Ankara, 06533 Turkey. Available at <http://bmi.os.u.edu/umit/software.htm>, 1999.
- [5] Ü. V. Çatalyürek, C. Aykanat, and B. Uçar, “On two-dimensional sparse matrix partitioning: Models, methods, and a recipe,” *SIAM J. Sci. Comput.*, vol. 32, no. 2, pp. 656–683, 2010.
- [6] Ü. V. Çatalyürek and C. Aykanat, “A fine-grain hypergraph model for 2D decomposition of sparse matrices,” in *Proc. 15th IPDPS*, San Francisco, CA, 2001, p. 118.
- [7] Ü. V. Çatalyürek and C. Aykanat, “A hypergraph-partitioning approach for coarse-grain decomposition,” in *Proc. Supercomputing, ACM/IEEE 2001 Conference*, Denver, Colorado, 2001, p. 42.
- [8] D. Chakrabarti, Y. Zhan, and C. Faloutsos, “R-mat: A recursive model for graph mining,” in *Proc. SDM*, 2004.
- [9] T. A. Davis and Y. Hu, “The University of Florida sparse matrix collection,” *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, 2011.
- [10] A. L. Dulmage and N. S. Mendelsohn, “Coverings of Bipartite Graphs,” *Canadian Journal of Mathematics*, no. 10, pp. 517–534, 1958.
- [11] K. Kaya, B. Uçar, and Ü. V. Çatalyürek, “Analysis of partitioning models and metrics in parallel sparse matrix-vector multiplication,” in *Proc. PPAM*, Warsaw, Poland, 2014, pp. 174–184.
- [12] V. Kuhlmann and P. S. Vassilevski, “Improving the communication pattern in matrix-vector operations for large scale-free graphs by disaggregation,” *SIAM J. Sci. Comput.*, vol. 35, no. 5, pp. S465–S486, 2013.
- [13] J. Leskovec and R. Sosič, “SNAP: A general purpose network analysis and graph mining library in C++,” <http://snap.stanford.edu/snap>, 2014.
- [14] D. M. Pelt and R. H. Bisseling, “A medium-grain method for fast 2D bipartitioning of sparse matrices,” in *Proc. 28th IPDPS*, Phoenix, Arizona, May 2014, pp. 529–539.
- [15] A. Pothén and C.-J. Fan, “Computing the block triangular form of a sparse matrix,” *ACM Trans. Math. Softw.*, vol. 16, no. 4, pp. 303–324, 1990.
- [16] R. O. Selvitopi, M. Ozdal, and C. Aykanat, “A novel method for scaling iterative solvers: Avoiding latency overhead of parallel sparse-matrix vector multiplies,” *IEEE T. Parall. Distr.*, vol. PP, no. 99, pp. 1–1, 2014.
- [17] B. Uçar and C. Aykanat, “Revisiting hypergraph models for sparse matrix partitioning,” *SIAM Rev.*, vol. 49, no. 4, pp. 595–603, 2007.
- [18] B. Vastenhouw and R. H. Bisseling, “A two-dimensional data distribution method for parallel sparse matrix-vector multiplication,” *SIAM Rev.*, vol. 47, no. 1, pp. 67–95, 2005.
- [19] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, “Graphx: A resilient distributed graph system on spark,” in *First International Workshop on Graph Data Management Experiences and Systems*, ser. GRADES ’13. New York, NY, USA: ACM, 2013, pp. 2:1–2:6.
- [20] A. Yoo, A. H. Baker, R. Pearce, and V. E. Henson, “A scalable eigensolver for large scale-free graphs using 2D graph partitioning,” in *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, pp. 63:1–63:11.
- [21] A. Yoo, E. Chow, K. Henderson, W. McLendon, B. Hendrickson, and Ü. V. Çatalyürek, “A scalable distributed parallel breadth-first search algorithm on BlueGene/L,” in *Proc. 2005 ACM/IEEE Conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2005, p. 25.